

UNA MIRADA AL AFINAMIENTO DE POSTGRESQL

JUAN CARLOS GUTIERREZ MARTINEZ
CODIGO 10 111 700

ADMINISTRACIÓN DE SISTEMAS DE INFORMACIÓN
INGENIERO ANGEL AUGUSTO AGUDELO ZAPATA

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
2009

INTRODUCCION

PostgreSQL, es un sistema gestor de base de datos, orientada a objetos de software libre, publicado bajo la licencia BSD.

El proceso de afinar una base de datos para mejorar el rendimiento se logra con el mejor aprovechamiento de los recursos de memoria, acceso a disco y comunicación en red.

Por lo general, cuando se instala PostgreSQL, este se deja con los valor por defecto. Cambiar algunos de estos valores lleva a conseguir un mayor rendimiento sin entrar a modificar las aplicaciones.

En este documento se presentan una buenas prácticas, tomadas del documento "PostgreSQL, afinamiento de la base de datos", disponible en Internet: http://wiki.postgresql.org/images/c/c5/Afinamiento_de_la_base_de_datos.pdf . Se han organizado los temas para mejor comprensión.

Se ha tenido como referencia el tutorial: "PostgreSQL Performance Tuning", disponible en Internet en: <http://linuxfinances.info/info/postgresqlperformance.html>

1. CONSIDERACIONES SOBRE EL SISTEMA DE DISCOS

El sistema de discos es uno de los cuellos de botella mas comunes en un sistema de base de datos.

Aunque aumentar la memoria puede mejorar la situación para bases de datos pequeñas, no siempre es la solución

Configuración RAID nivel 10 es lo que la práctica recomienda.

Otra buena práctica es pagar cache de escritura o usar battery backed array controllers.

Se recomienda configurar el WAL (\$PGDATADIR/pg_xlog) en un disco aparte.

Si es posible, disponer de discos separados para poner los datos, índices y archivos temporales.

2. Configuraciones de Postgresql.conf relevantes al rendimiento.

2.1 CONTROLANDO EL USO DE LOS RECURSOS

2.1.1 "*max_connections*"

2.1.1.1 Máximo número de conexiones permitidas a la base de datos.

2.1.1.2 Debe mantenerse en un valor razonable.

2.1.1.3 Al escoger el valor de max_connections se debe considerar el uso de work_mem.

2.1.1.4 Se puede usar algún pool de conexiones (PgPool, PgBouncer) para mantener este valor bajo.

2.1.2 "*shared_buffers*"

2.1.2.1 Este parámetro determina cuanta memoria puede usar PostgreSQL de forma dedicada para mantener datos en cache.

2.1.2.2 Un valor razonable (para empezar a probar) es el 25% del total de la memoria RAM disponible.

2.1.2.3 Si no se indica la unidad se asume 8Kb por cada shared buffer. (shared_buffers = 1024 = 8Mb)

2.1.2.4 Puede necesitarse alterar el valor de SHMMAX.

2.1.3 "work_mem"

2.1.3.1 Este parámetro determina cuanta memoria puede usar PostgreSQL en operaciones de ordenamiento y para tablas hash antes de usar archivos temporales.

2.1.3.2 Este valor se aplica por operación (una consulta realizando varios ordenamientos usará X veces este valor).

2.1.3.3 Al escoger el valor de work_mem se debe considerar el valor de max_connections, un valor razonable puede ser entre el 2-4% del total de la memoria.

2.1.3.4 Para mitigar el costo de usar archivos temporales se puede usar el parámetro temp_tablespace.

2.1.4 "max_fsm_pages, max_fsm_relations"

2.1.4.1 Guardan información sobre el espacio libre (reutilizable) generado debido a la existencia de tuplas muertas en la base de datos.

2.1.4.2 El FSM solo se registra durante vacuum

2.1.4.3 **`max_fsm_relations`** debería ser igual al total de tablas e índices en todas las bases de datos de una instalación.

2.1.4.4 Se puede usar **`vacuum verbose`** para determinar el valor de **`max_fsm_pages`**.

2.2 Mitigando el costo de escritura a disco.

2.2.1 **`fsync`, `synchronous_commit`**

2.2.1.1 Determina si todas las páginas del WAL deben grabarse a disco antes de que se considere terminada la transacción.

2.2.1.2 Configurar este valor a OFF podría causar corrupción de datos ante una eventual caída del servidor. En instalaciones de Data Warehouse podría apagarse para aumentar el rendimiento.

2.2.1.3 `synchronous_commit` puede proveer la mejora en el rendimiento que `fsync=off` daría sin el riesgo de corrupción de datos. Este parámetro se puede configurar en cualquier momento.

2.2.2 **`commit_delay`, `commit_siblings`**

2.2.2.1 Se usan para tratar de ejecutar COMMIT simultáneamente en varias transacciones.

2.2.2.2 Si existen otros backends activos cuando una transacción está haciendo COMMIT, el servidor espera `commit_delay` microsegundos con la esperanza de que alguna otra de las transacciones (hasta que la operación `commit_siblings`) termine

2.2.3 checkpoints

`checkpoints_segments`
`checkpoints_timeout`
`checkpoints_completion_target`

2.2.3.1 Pueden ocurrir en 2 circunstancias:

La primera: Se han llenado todos los segmentos del WAL (`checkpoint_segment`)

La segunda Ha transcurrido `checkpoint_timeout` segundos desde el último checkpoint.

2.2.3.2 `checkpoint_completion_target` fue concebido para distribuir uniformemente la ejecución del checkpoint actual durante el período de espera al siguiente.

2.3 Indicando a PostgreSQL como escoger los mejores planes.

2.3.1 random_page_cost

2.3.1.1 Determina la forma en que el planeador considera los accesos no secuenciales a disco. Un valor bajo favorecerá el uso de índices; un valor alto, las lecturas secuenciales.

2.3.1.2 Aunque es prudente reducir un poco este valor, debe recordarse que el uso de índices tiene un costo y nunca será más rápido que el acceso secuencial(seq_page_cost).

2.3.2 effective_cache_size

2.3.2.1 Determina la estimación del planeador en cuanto a cuanta memoria hay disponible para mantener un cache para las consultas. Un valor alto favorecerá el uso de índices; un valor bajo, las lecturas secuenciales.

2.3.2.2 Se debería configurar al valor de la memoria que queda luego de considerar shared_buffers, Sistema Operativo y otras aplicaciones.

2.3.2.3 Un valor razonable, dependiendo de la instalación, puede ser de 1/2 del total de la memoria.

2.3.3 constraint_exclusion

2.3.3.1 Permite al planeador considerar los constraints checks en la tabla al determinar el plan de ejecución. Se usa cuando se esta usando un esquema de partición basado en herencia de tablas

2.3.3.1 En el siguiente ejemplo solo se accede a la tabla child2000.

```
CREATE TABLE parent(key integer, ...);
CREATE TABLE child1000(
check (key between 1000 and 1999)
) INHERITS(parent);
CREATE TABLE child2000(
check (key between 2000 and 2999)
) INHERITS(parent);
...
SELECT * FROM parent WHERE key = 2400;
```